

Functional Programming

and Monad

yagom

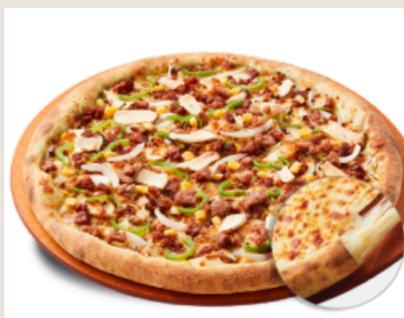
피자 만들기 프로젝트



스파이시쉬림프 크러스트

살이 통통한 케이준 새우와 매콤한 피칸테소스의 환상적인 조화!

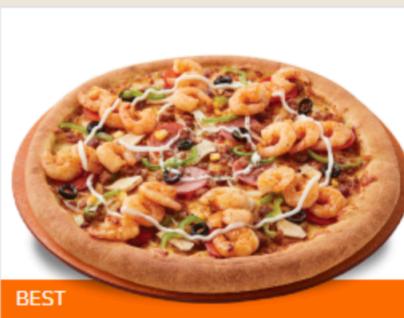
M: 19,900 원 / L: 22,900 원



불고기 크러스트

한국인의 입맛을 사로잡는 불고기와 피자의 기막힌 궁합!

M: 17,900 원 / L: 20,900 원



스파이시쉬림프

살이 통통한 케이준 새우와 매콤한 피칸테소스의 환상적인 조화!

M: 16,900 원 / L: 19,900 원



슈프림콤비

피자의 기본중의 기본! 다양한 재료들이 한판에 가득가득 기막힌 콤비네이션!

M: 13,900 원 / L: 16,900 원



BEST

불고기

한국인의 입맛을 사로잡는 불고기와 피자의 기막힌 궁합!

M: 14,900 원 / L: 17,900 원



리치킨BBQ

부드러운 닭가슴살 치킨텐더와 매콤한 피칸테소스, 달콤한 고구마무스의 매콤달콤한 만…

M: 16,900 원 / L: 19,900 원



토핑들의수다

한국인이 가장 좋아하는 육,해,공 디저트 대표 재료가 한판에!

M: 20,900 원 / L: 24,900 원



웨지포테이토 소보로

담백하고 포슬포슬한 웨지감자와 크리미한 특제마요소스의 만남!

M: 17,900 원 / L: 20,900 원

뿅뜨락피자 메뉴 일부 발췌

피자 만들기 프로젝트

무엇부터 시작하면 좋을까요?

- 피자 완성 결과물을 보여줄 서비스
- 피자 레시피

| | 콤비네이션 피자 | 불고기 피자 | 치즈 피자 |
|----|-------------|-----------|----------|
| 도우 | 밀가루 | 쌀 | 밀가루 |
| 소스 | 바베큐 | 바베큐 | 토마토 |
| 토핑 | 버섯 | 불고기 | 치즈 |

in OOP

```
class 피자 {  
    var 도우  
    var 소스  
    var 토피  
}
```

피자 콤비네이션피자 = 피자(밀가루, 바베큐, 버섯)

피자 불고기피자 = 피자(쌀, 바베큐, 불고기)

피자 치즈피자 = 피자(밀가루, 토마토, 치즈)

in Functional Programming

반죽 밀가루반죽 = 반죽하기(밀가루)

도우 불고기피자도우 = 바르기(밀가루도우, 바베큐소스)

굽기전 불고기피자굽기전 = 얹기(불고기피자도우, 불고기)

피자 불고기피자 = 굽기(불고기피자굽기전)

반죽(밀가루).바르기(바베큐소스).얹기(불고기).굽기()

Currying

커링

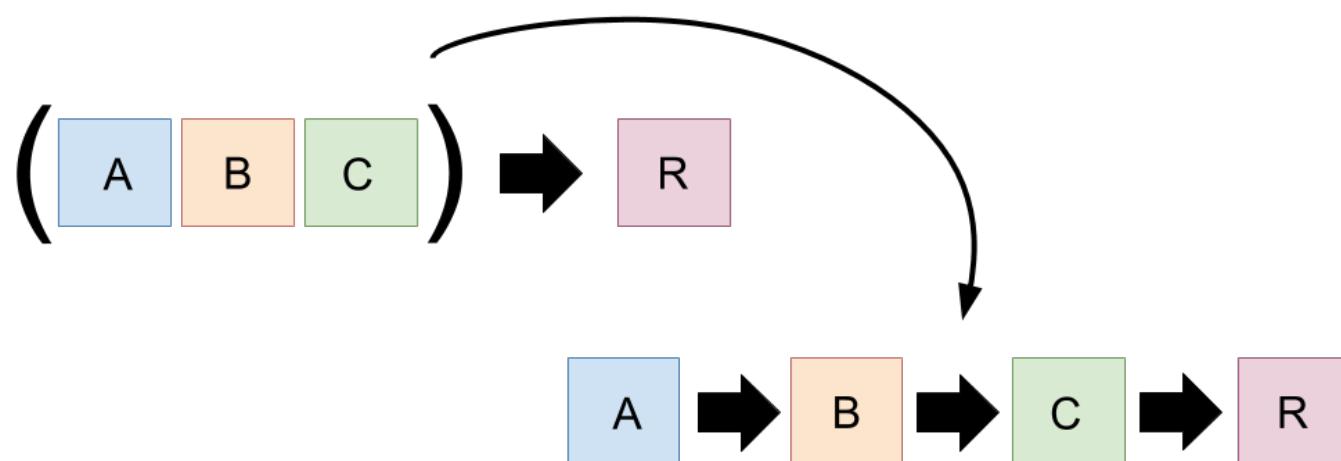
- 하스켈 커리(Haskell Brooks Curry)
- 여러개의 매개변수를 갖는 함수를 하나의 매개변수를 갖는 함수로 변환하는 것
 - $(A,B) \rightarrow B$ 보다는 $A \rightarrow (B \rightarrow B)$ 로!
 - 함수를 만드는 함수
- 커링의 장점
 - 함수의 재사용성 향상
 - 매개변수 중 변하지 않는 값은 고정시키고
변하는 값만 변경하면서 함수를 실행하도록 구현
 - 코드의 가독성 향상



Currying

커링

- 여러개의 매개변수를 갖는 함수를
하나의 매개변수를 갖는 함수로 변환하는 것
- 변환된 함수는 매개변수 하나를 입력받고 값 대신 함수를 반환
이 반환된 함수는 다음 매개변수를 입력으로 받고,
입력 값을 모두 처리하고 반환하는 값이 하나만 남을 때까지 이 과정을
반복



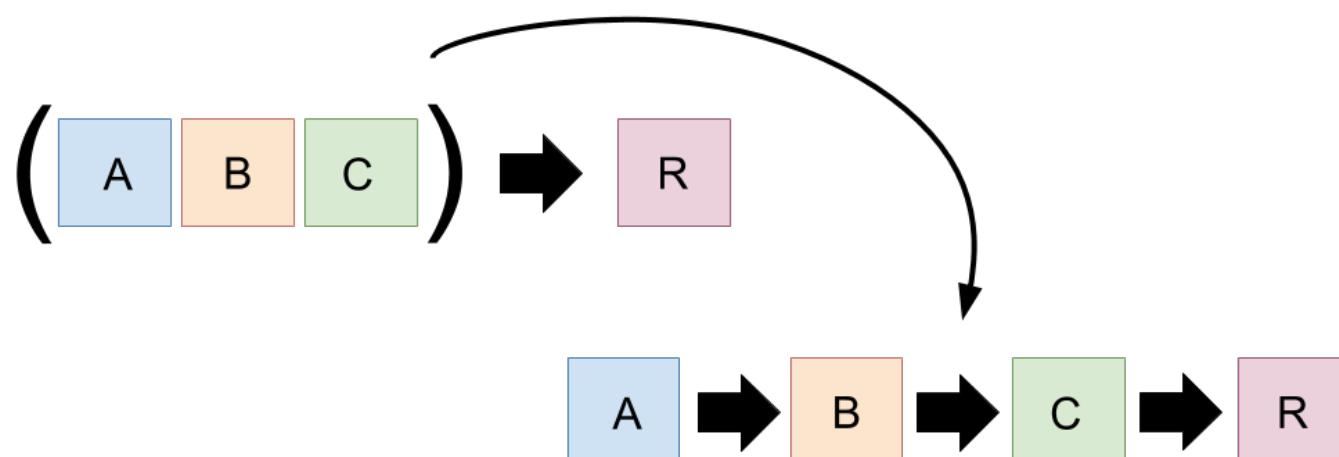
출처 : 토미의 개발노트

Currying

커링

피자(밀가루, 바베큐소스, 불고기)
피자(쌀가루, 토마토소스, 치즈)

피자(밀가루)(바베큐소스)(불고기)
피자(쌀가루)(토마토소스)(치즈)



출처 : 토미의 개발노트

속성시간을
옵션으로 가져야 한다면...?

Currying

커링

우리가 알고 있던 함수라면...

func 피자(반죽타입, 소스타입, 토픽타입) { ... }

피자(밀가루, 바베큐소스, 불고기)

피자(쌀가루, 토마토소스, 치즈)

func 피자(반죽타입, 숙성시간, 소스타입, 토픽타입) { ... }

피자(밀가루, 1시간, 바베큐소스, 불고기)

피자(쌀가루, 3시간, 토마토소스, 치즈)

커링 가능하다면...?

피자(쌀가루)(토마토소스)(치즈)

피자(쌀가루)(1시간)(토마토소스)(치즈)

Higher-order Functions

고차함수, 고계함수... 뭐... 번역은 지들망대로...

- 함수를 매개변수로 전달받는 함수

- 대표적인 예
 - filter
 - map
 - reduce

map, filter, and reduce
explained with emoji 😂

```
map([🍔, 🍟, 🍗, 🥗], cook)  
=> [🍔, 🍟, 🍗, 🥗]
```

```
filter([🍔, 🍟, 🍗, 🥗], isVegetarian)  
=> [🍟, 🥗]
```

```
reduce([🍔, 🍟, 🍗, 🥗], eat)  
=> 💩
```

Addy Osmani's facebook post

filter

Filter

```
let numbers: [Int] = [0, 1, 2, 3, 4, 5]
var evenNumbers: [Int] = [Int]()

// for 구문 사용
for number in numbers {
    if number % 2 != 0 { continue }
    evenNumbers.append(number)
}

print(evenNumbers) // [0, 2, 4]
```

FILTER EXAMPLE

```
let numbers: [Int] = [0, 1, 2, 3, 4, 5]

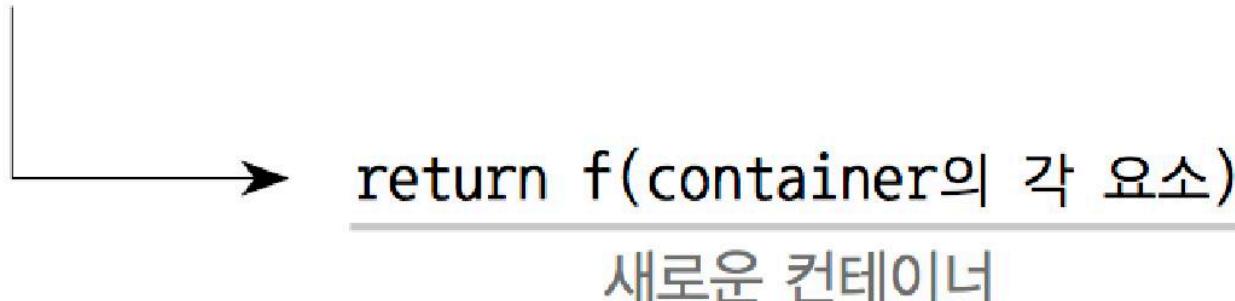
let evenNumbers: [Int] = numbers.filter { (number: Int)
-> Bool in
    return number % 2 == 0
}
print(evenNumbers) // [0, 2, 4]
```

map

Transform

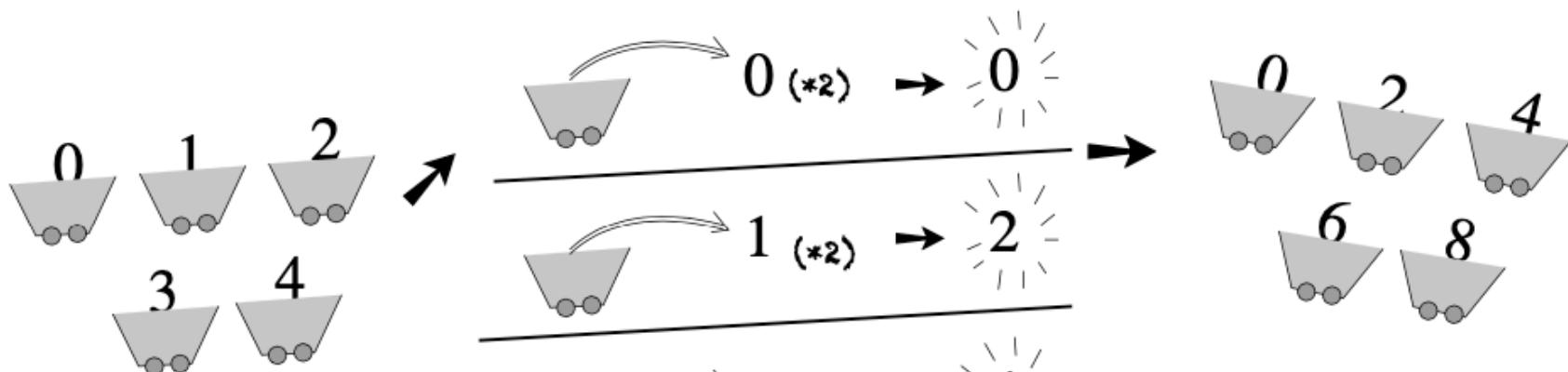
- 콜렉션 내부의 기존 데이터를 변형(transform)하여 새로운 콜렉션 생성

`container.map(f(x))` 컨테이너의 map 메서드 호출



```
map([🐮, 🍟, 🐔, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]
```

map Transform



1. numbers Array

3. 클로저의 연산 결과값을 담은
새 Array doubledNumbers

2. numbers Array의 각 요소에 map
메서드의 전달인자로 전달한 클로저 적용

TRANSFORM USING FOR STATEMENT

```
let numbers: [Int] = [0, 1, 2, 3, 4]
var doubledNumbers: [Int] = [Int]()
var strings: [String] = [String]()

// for 구문 사용
for number in numbers {
    doubledNumbers.append(number * 2)
    strings.append("\(number)")
}

print(doubledNumbers) // [0, 2, 4, 6, 8]
print(strings) // ["0", "1", "2", "3", "4"]
```

MAP EXAMPLE

```
// map 메서드 사용
doubledNumbers = numbers.map({ (number: Int) -> Int in
    return number * 2
})

strings = numbers.map({ (number: Int) -> String in
    return "\u2022(number)"
})

print(doubledNumbers) // [0, 2, 4, 6, 8]
print(strings) // ["0", "1", "2", "3", "4"]
```

reduce(fold)

```
let numbers: [Int] = [2, 8, 15]
var sum: Int = 0

for number in numbers {
    sum += number
}

print(sum) // 25
```

REDUCE EXAMPLE

```
let numbers: [Int] = [2, 8, 15]

// 초기값이 0이고 정수 배열의 모든 값을 뺍니다.
let subtract: Int = numbers.reduce(0, { (first: Int,
second: Int) -> Int in
    print("\(first) - \(second)")
    return first - second
})

print(subtract) // -25
```

Functor

함수객체, 함자, 평터...

- map을 적용할 수 있는 컨테이너 타입
- Array(List), Dictionary(Hash map), Set 등등..

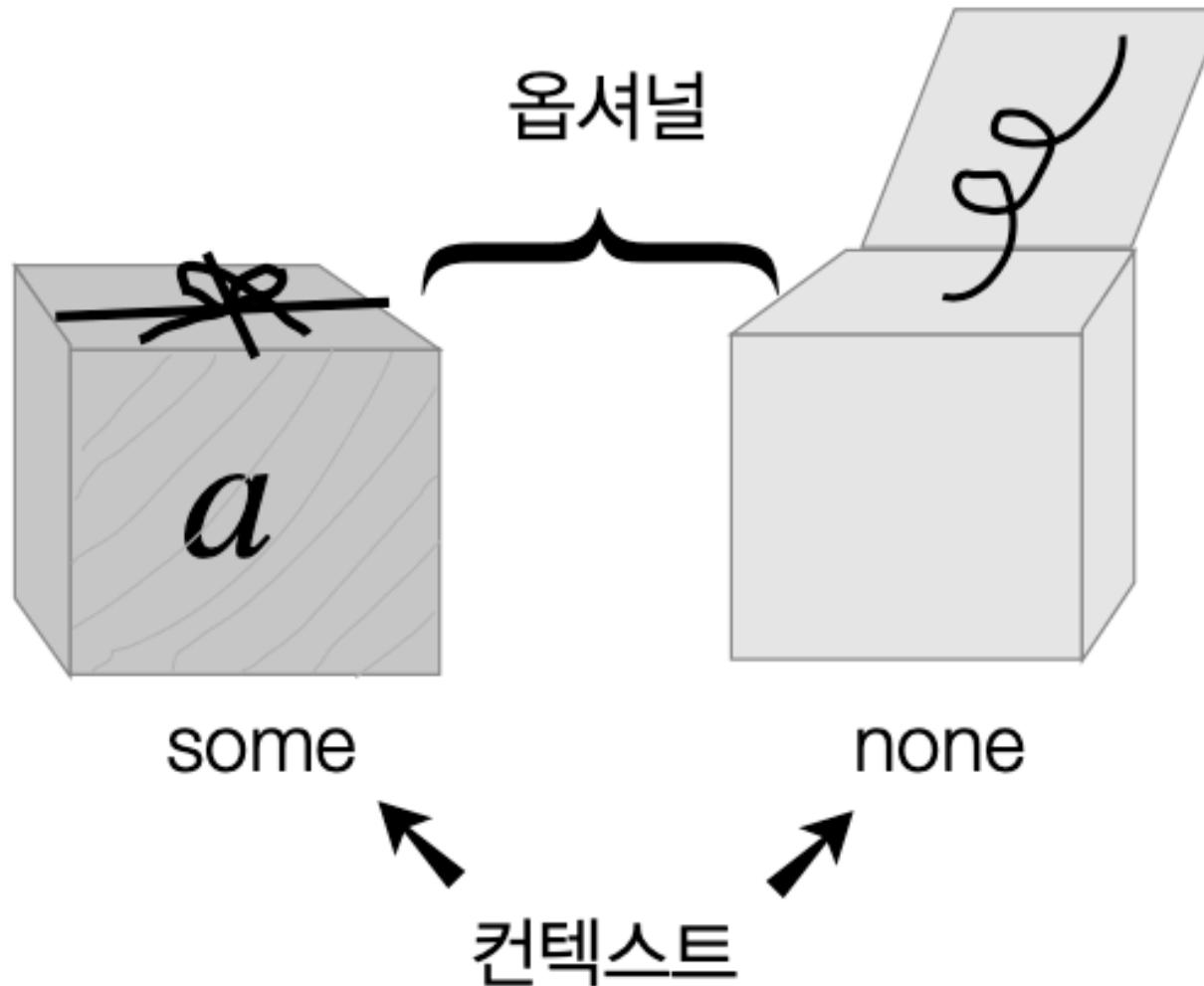
Monad

모나드, 몰라 이제 나도...

- 컨텍스트에 포장된 값을 처리하여 포장된 값을 같은 컨텍스트에 다시 담아 반환하는 함수객체
- 쉽게 말해서... (날림임, 이게 항상 적용되는 것이 아님)
(map을 적용할 수 있는) List라는 타입이, 자신의 각각의 값에 변환 처리를 한 후(mapping) 다시 List로 반환한다면 List는 모나드이다.

Optional

옵션



Method Chainig

```
결과 = 친구목록.filter { 나이 > 20 }
    .filter { 성별 == 여 }
    .map { 성을 대문자로 변환하는 함수 }
    .map { 힌티셔츠 입기 함수 }
```

Pure Functions

순수함수

- $y = x + 3$
- func 반죽하기(재료) => 반죽
 - 반죽하기(밀가루) => 밀가루반죽
 - 반죽하기(쌀가루) => 쌀가루반죽
- func 굽기(재료) => 구이
 - 굽기(오리) => 오리구이
 - 굽기(옥수수) => 군옥수수

REDUCE EXAMPLE

```
let numbers: [Int] = [2, 8, 15]
// 초기값은 0이고 정수 배열의 모든 값을 더합니다.
let sum: Int = numbers.reduce(0, { (first: Int, second: Int) -> Int in
    print("\(first) + \(second)")
    return first + second
})

print(sum) // 25
```



Benefit of Functional Programming

- 대규모 병렬처리에 유리
- 더 큰 규모의 재사용성
- 더 큰 최적화 가능성

OOP+Functional

- 피자만 먹고 살 수 없다. 떡볶이도 먹고, 김피탕도 먹자.
- 커링까지 진국을 먹진 않더라도 체이닝 정도의 멋짐은 부려보자

```
public static void main(String[] args)
{
    // Create User
    User user = User.Builder()
        .id("123")
        .name("Test")
        .address("In the Building")
        .age(20)
        .createUser();

    // User created!

    System.out.println(user);
}
```